# CMPSC265 Data Structures and Algorithms
# Homework 4

**Due: Sunday, Sept 29nd, 2019, 11:59pm**

**Learning Objectives**

- Applications on Stack
- Applications on Queue
- Applications on Priority Queue

**Deliverables:**

**A zipped folder named as *YOURLASTNAME.zip* containing all the following java files:**

- The modified java file *InfixEvaluator.java* for Problem 1
- The modified java file *Josephus.java* for Problem 2
- The modified java file *MovingAverage.java* for Problem 3
- The modified java file *Ramanujan.java* for Problem 4

**Instructions:**

- Please first download the Java source file for all four problems, modify the codes based upon requirements, and submit the updated source file.
- Please also download the Java source file for QueueX.java for the implementation for the Queue data type, and save it in the same package for your other java files for problem 2-3. You will need it for your implementation of Problem 2 to Problem 3. We are not using the Java built-in class for Queue.
- For Problem 1, please use the Java built-in class for Stack.
- For problem 4 on PriorityQueue, please use the Java built-in class for PriorityQueue.
- Please add the required *credit comments* and *comments* to each Java program you submit.

**Problem Specifications:**

**Problem 1: InfixEvaluator class (25')**

**Description:**

Please write a program that can evaluate an infix arithmetic expressions involving doubles combined with +, -, *, /, and ^ operators as well as

parenthesis. The parenthesis does NOT have to be fully balanced. For example:
( 2 * 3 ^ 2 ). The fully parenthesized version of this expression would be ( 2 * ( 3 ^ 2 ) )
However, make sure that:

- Each such expression always has one opening parenthesis and one closing parenthesis. For example, you need to represent as ( 2 + 3 ) not 2 + 3.
- Numbers and operators including parenthesis are all separated by **a space**.
- Please do not first convert the infix expression into postfix, and then evaluate the postfix.

**Hints: You need to have two stacks for this problem.** One stack is to hold all operators, and the other is to hold numbers. Here shows a working algorithm for you reference:

1. While there are still tokens to be read:
   A. Get the next token.
   B. If the token is:
      a) A number: push it onto the value stack.
      b) A left paren "(": push it onto the operator stack.
      c) A right paren ")":
         1) While the top element of the operator stack **(use peek() not pop())** is not a left paren:
            - Pop the operator from the operator stack.
            - Pop the value stack twice, getting two operands.
            - Apply the operator to the operands, in the correct order.
            - Push the result onto the value stack.
         2) Pop the left parenthesis from the operator stack, and discard it.
      d) An operator:
         1) While the operator stack is not empty, and the top thing on the operator stack has the same or greater **precedence** as the token,
            - Pop the operator from the operator stack.
            - Pop the value stack twice, getting two operands.
            - Apply the operator to the operands, in the correct order.
            - Push the result onto the value stack.
         2) Push the token onto the operator stack.
2. At this point the operator stack should be empty, and the value stack should have only one value in it, which is the final result. Pop it and print it to System.out.

**Outputs:**

Your output should look something like follows.

```
$ java InfixEvaluator
Please enter an arithmetic expression: ( 2 + 3 * 4 ^ 2)
The result is: 50.0

$ java InfixEvaluator
Please enter an arithmetic expression: ( 10 – 3 * 8 / –4 )
The result is: 16.0
```

**Problem 2:** ***Josephus class (25')***

**Description:**

In the Josephus problem from antiquity, N people are in dire straits and agree to the following strategy to reduce the population. They arrange themselves in a circle (at positions numbered from 0 to N − 1) and proceed around the circle, eliminating every Mth person until only one person is left. Legend has it that Josephus figured out where to sit to avoid being eliminated.

Finish the Queue client Josephus.java that takes N and M from keyboard and prints out the order in which people are eliminated (and thus would show Josephus where to sit in the circle).

**Outputs:**

Your outputs should look something as follows.

```
$ java Josephus
Please enter a number for N indicating the total number of
people: 7
Please enter a number M indicating the Mth person to be
eliminated: 2
People will be eliminated in such a sequence:
1 3 5 0 4 2 6

$ java Josephus
Please enter a number for N indicating the total number of
people: 20
Please enter a number M indicating the Mth person to be
eliminated: 3
People will be eliminated in such a sequence:
2 5 8 11 14 17 0 4 9 13 18 3 10 16 6 15 7 1 12 19
```

**Problem 3: MovingAverage class (25')**

**Description:**

Write a class that can implement the following: Given a stream of integers and a window size, calculate the moving average of all integers in the sliding window.

MovingAverage m = new MovingAverage(3);
m.next(1) = 1
m.next(10) = (1 + 10) / 2
m.next(3) = (1 + 10 + 3) / 3
m.next(5) = (10 + 3 + 5) / 3

**Outputs:**

Your outputs should look something as follows.

```
$ java MovingAverage
Please enter a number to specify the size of the window: 3
Please enter a number to be added into the window: 1
The average after adding the above number is: 1.0
Please enter a number to be added into the window: 10
The average after adding the above number is: 5.5
Please enter a number to be added into the window: 3
The average after adding the above number is: 6.0
```

**Problem 4: Ramanujan class (25')**

**Description**

Srinivasa Ramanujan was an Indian mathematician who became famous for his intuition for numbers. When the English mathematician G. H. Hardy came to visit him one day, Hardy remarked that the number of his taxi was 1729, a rather dull number. To which Ramanujan replied, "No, Hardy! It is a very interesting number. It is the smallest number expressible as the sum of two cubes in two different ways. Since:
1729 = 1^3 + 12^3 = 9^3 + 10^3

Please finish writing the program Ramanujan.java using a minimum-oriented PriorityQueue to find all such numbers that are less than or equal to N and that each can be expressed as the sum of two cubes in two different ways. In other words, find distinct positive integers *i, j, k, and l* such that $i^3 + j^3 = k^3 + l^3$.

Algorithm is working as follows:

1) Initialize a minimum-oriented priority queue with pairs (1,2), (2,3), (3,4), ..., (i, i + 1) such that $i < \sqrt[3]{N}$.
2) While the priority queue is nonempty:

- remove the pair (i, j) such that $i^3 + j^3$ is the smallest (this is easily done since we are using a minimum-oriented priority queue);
- print the previous pair (k, l) and the current pair (i, j), if $k^3 + l^3 = i^3 + j^3 \leq N$,
- and then if $j < \sqrt[3]{N}$, insert (i, j+1) into the priority queue.

**Outputs:** Your results should look similar as follows.

$ java Ramanujan

```
Please enter a number: 40000
1729 = 1^3 + 12^3 = 9^3 + 10^3
4104 = 2^3 + 16^3 = 9^3 + 15^3
13832 = 18^3 + 20^3 = 2^3 + 24^3
20683 = 19^3 + 24^3 = 10^3 + 27^3
32832 = 18^3 + 30^3 = 4^3 + 32^3
39312 = 15^3 + 33^3 = 2^3 + 34^3
```